

QAFE Boot Camp - Course Material

Environment Set-up

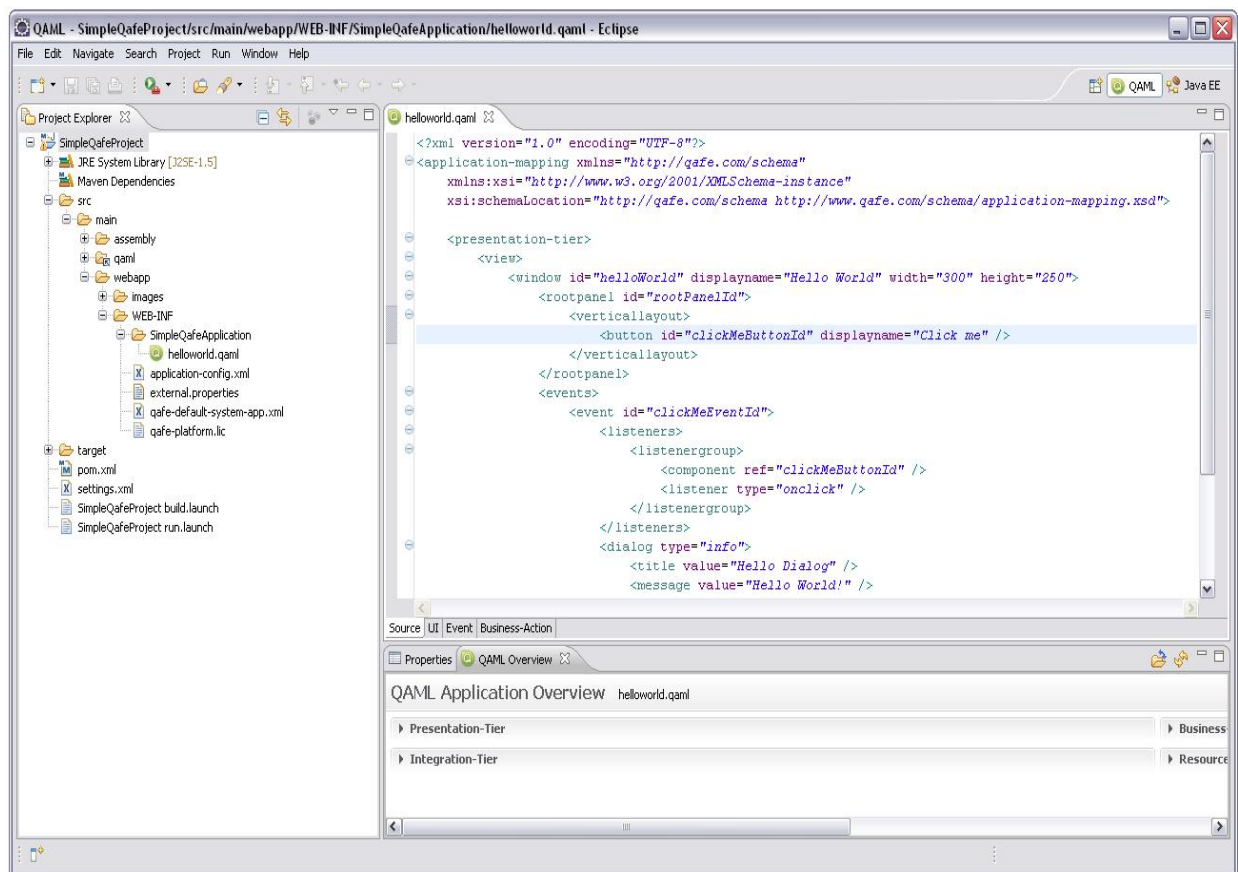
Get the latest QAFE software from www.qafe.com/download and follow the instructions

Create a Simple QAFE Project

Follow the set of instructions to set up a new QAFE project using the Eclipse IDE.

1. Open Eclipse.
2. Click OK and continue with the default choice of workspace.
3. Right click in the **Project Explorer** view, choose > **New** > **Project**.
4. In the **New Project** window, choose > **QAFE** > **QAFE Project**.
5. Click on **Next**.
6. Provide *SimpleQafeProject* as **Project Name**.
7. Provide *SimpleQafeApplication* as **Application Name**.
8. For **Path to QAFE**, browse and choose QAFE-Platform-xxxx directory from your <home>/qafe/
9. Click on **Finish**.
10. Right click on *SimpleQafeProject*, choose > **Maven** > **Update Project Configuration**.

Following screen shot represents the result of the above mentioned steps.

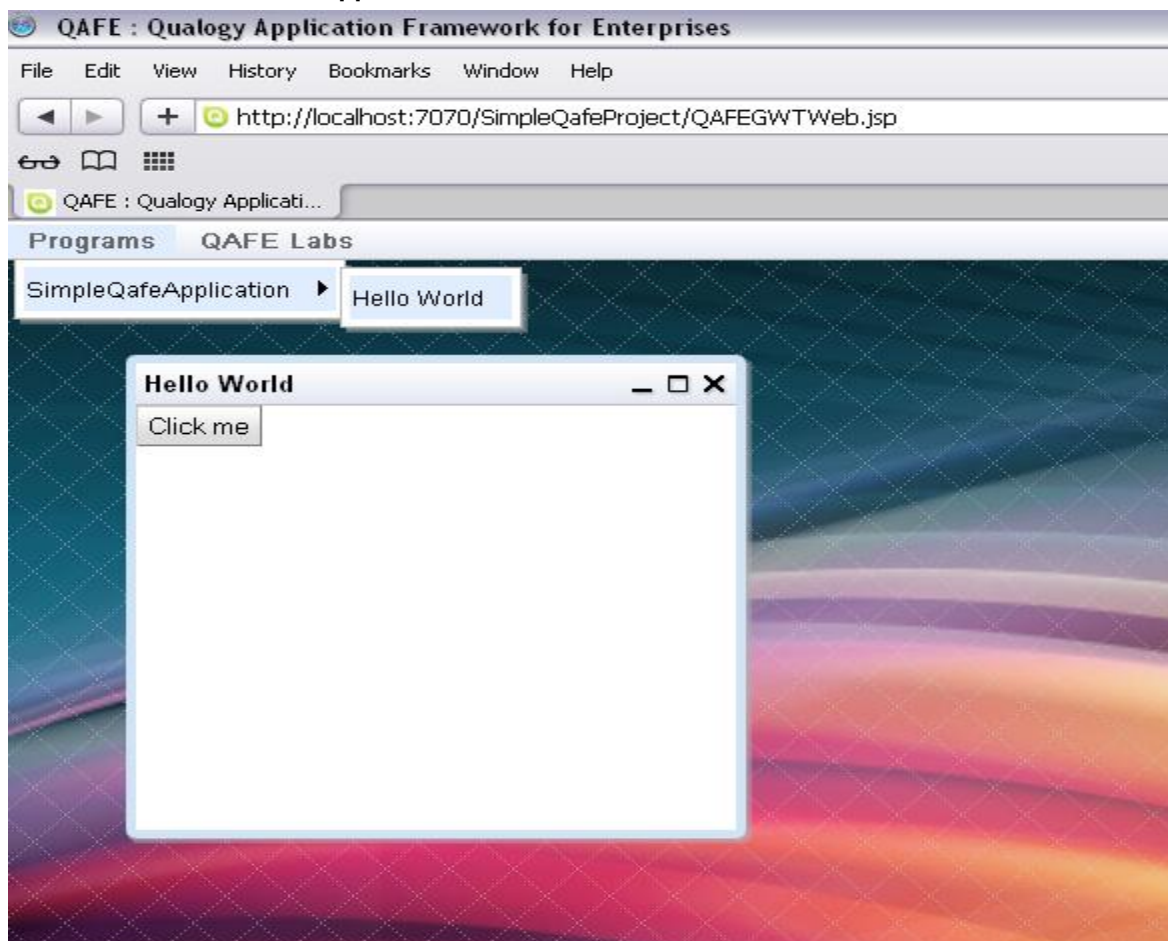


Build and Run a QAFE Application

- To build the project - Right click on *SimpleQafeProject* and choose > **Build QAFE Application**
 - In Eclipse the console will show log messages of the build. It will show Build Success message once the build is finished. Once the build is done successfully do the next step.
- To run the project - Right click on *SimpleQafeProject* and choose > **Run as > QAFE Application**.
 - The Eclipse console will show up log messages deploying the application in the built in jetty application server. The URL for the application will show up towards the end of the log once the application is deployed successfully.

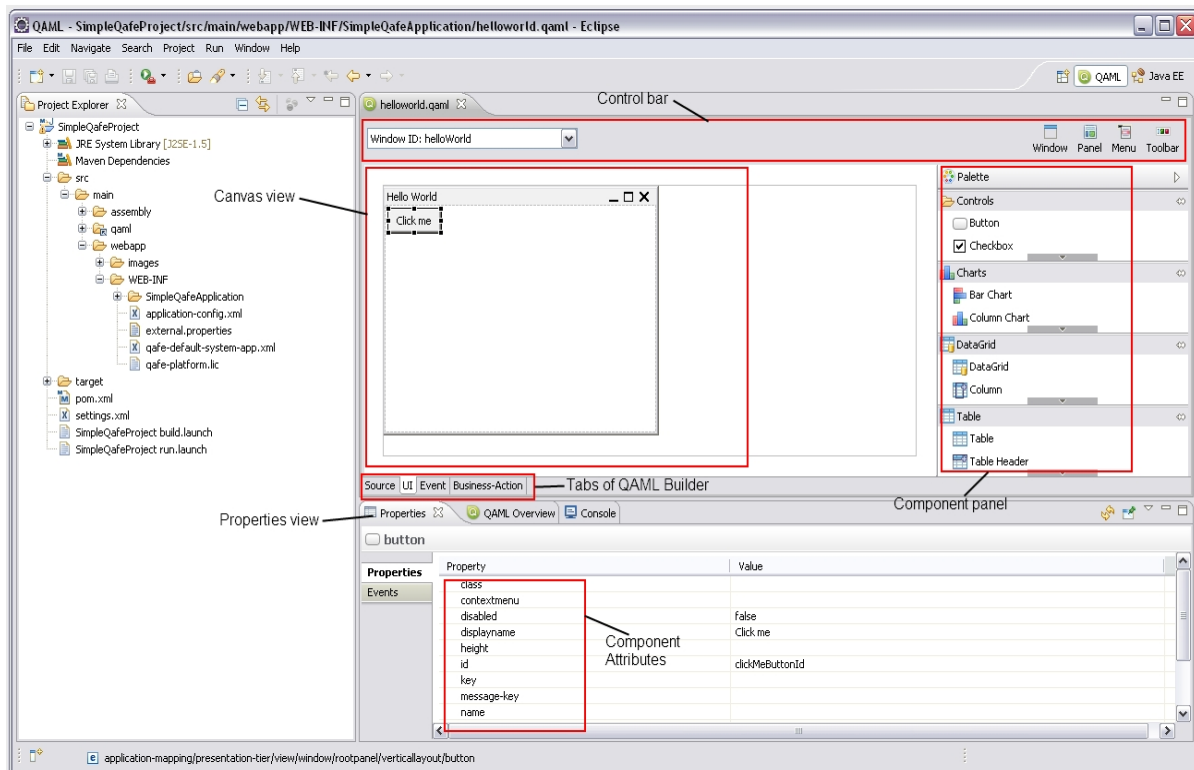
The default browser will start up with the QAFE Application. In the application invoke **Programs > SimpleQafeApplication > Hello World**.

From here on any change in the QAML code will be reflected in the browser after reloading. Use the menu item **QAFE Labs > Reload applications** to reload.



NOTE: In any case where the changes made on the project are not taking effect in the browser, rebuilding of the project is necessary. Before building the project the server should be stopped. After building the project run the project again to see the changes reflected in the application.

QAML Builder plug-in parts in Eclipse



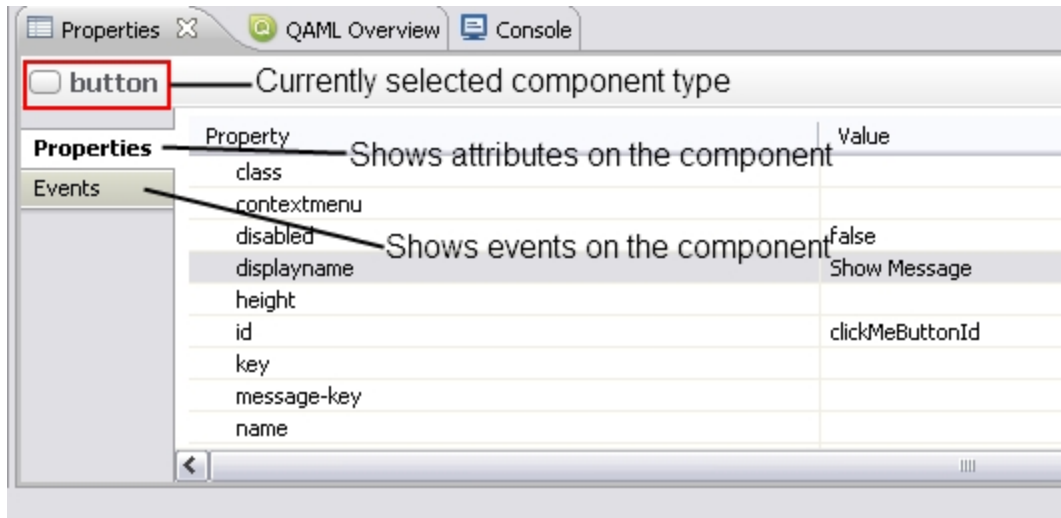
Exercise 1: Getting started with QAML Builder

This exercise will enable you to modify the helloworld.qaml file and to add some more components. The expected result is an application which shows a dialog box when a button is clicked with messages which are typed in a text field. The focus of this exercise is to understand

- How to create new components.
- How to associate an event to a component.
- How to access data of a component.

Steps:

1. In eclipse go to the **UI** tab
2. From the component palette drag'n drop a **Textfield** component before the existing button.
3. At the prompt type *Message To Show*. Hit Enter
 - The text will reflect as value to the **displayname** attribute in the properties view of the text field component.
 - **NOTE:** The display name can be edited in the canvas view by clicking on it.
4. Use the properties view and provide *messageInputTextField* as **id** of the text field.
5. Select the existing button component by clicking on it.
6. In the properties view change the **displayname** attribute to *Show Message*.

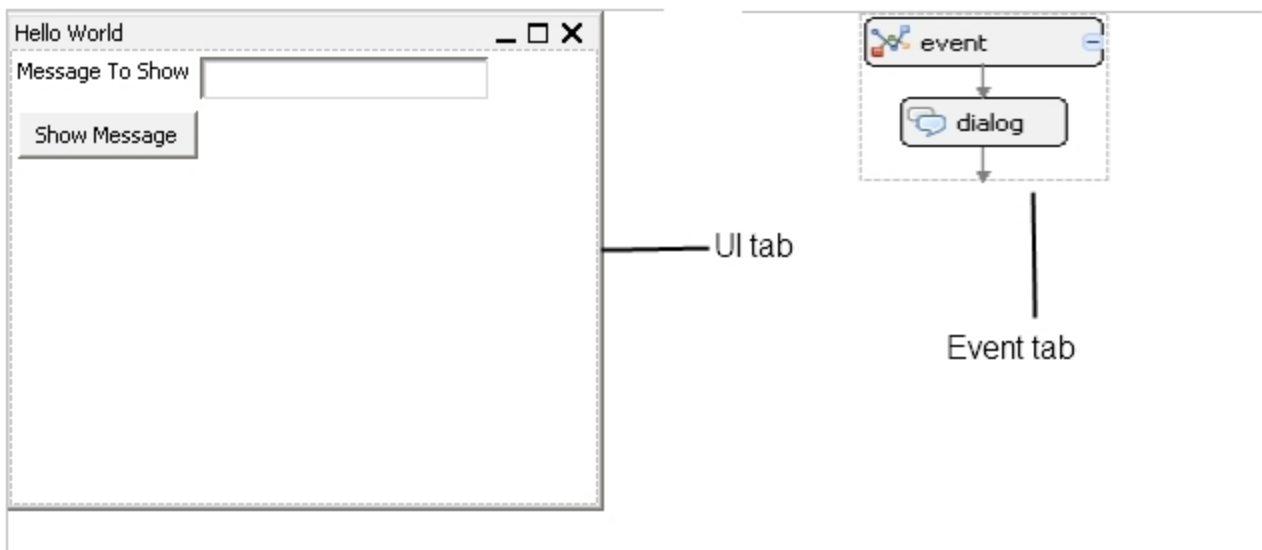


7. Click on Events in the Properties view.
8. Double click on the existing event.
 - Events associated with a component can be seen as list. Double click on an event from the list will take the focus to Events tab in the canvas view. The component palette will automatically reflect QAFE built ins which can be used inside an event.
9. In the canvas view select the dialog component.
10. In the properties view click on the text field next to the **message** attribute of the dialog component. A pop up window appears which manages attributes of the message to be shown in the dialog.
11. In the pop-up window choose *component* from the drop down next to the **src** attribute of message.
12. Choose *messageInputTextField* as **ref** attribute of the message.
13. Click **OK**.
14. Save the file, reload the browser and invoke the application again to test the changes made.

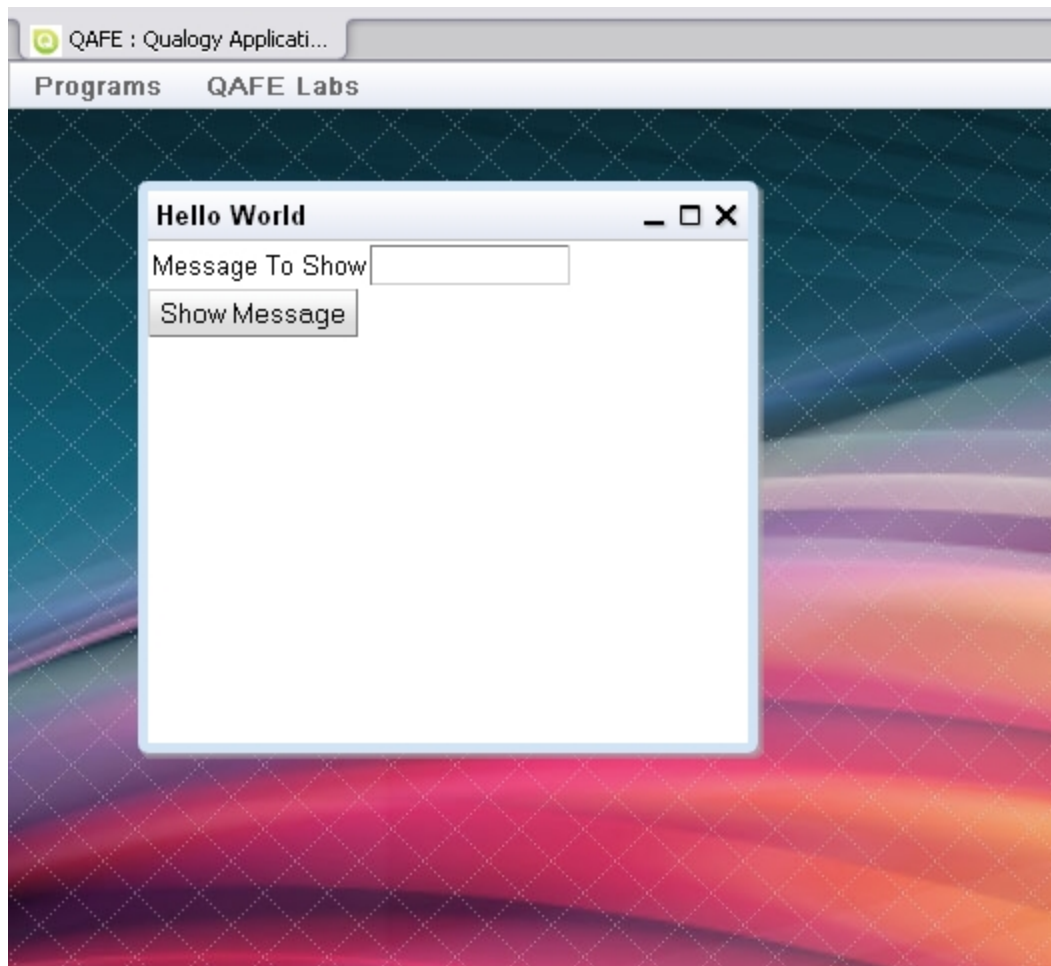
Things to understand from this exercise

- How to add components.
- How to add or edit attribute values of a selected component.
- How to identify events associated with a component.
- How to edit or add logic to events.
- How to reference a component using the id of the component.

After exercise 1 the UI tab and the event tab in eclipse will look as shown below:



Resulting application:



Exercise 2: Creating a QAFE application from scratch

This exercise will take you through steps to create a qaml file and to create and understand a window and rootpanel in the QAFE context. Furthermore we will learn

- How to create tabs as part of the UI in a qafe application.
- Get familiar with the layout of a panel.
- Using regular expressions to validate contents.
- Applying styles to components using CSS.

Steps:

1. In eclipse, go to *SimpleQafeProject* > *src* > *main* > *webapp* > *WEB-INF*.
2. Right click on *SimpleQafeApplication* folder > **New** > **Other** > **QAFE** > **Qaml File**.
3. Click on **Next**.
4. Provide *myPlaygroundApp* as name for the qaml file.
5. Click on **Finish**. The new qaml file will show up with focus on the **Source** tab.
6. Click on the **UI** tab.
7. Click on the **Window** component in the control bar. A pop-up will appear.
8. Provide *windowOne* as **id** and *Play Ground* as **displayname**.
9. Click on **OK**.
10. Select the new created window by clicking on it.
 - **NOTE:** Make sure that the click is performed on the title area of the window. Clicking inside the window will make the rootpanel component selected.
11. Optional: Drag and re-size the window to have a proportionate size with that of the canvas.
12. From the control palette drag'n drop a **TabPanel** component into the window.
 - The TabPanel component need to be considered as parent container for holding tab component(s) acting as child container to hold other UI components. Another container component called StackPanel has the same principle. For a StackPanel a Stack component acts as child component.
13. Drag'n drop a **Tab** component from the component palette into the newly created tab panel component.
14. Type *RegEx example* in the prompt for **displayname**.
15. Optional: Drag and re-size the tab to have a proportionate size with that of the window component
16. Drag'n drop a **Label** component into the tab.
17. Provide *Email-Id* as **displayname** for the label at the prompt..
18. Drag'n drop a textfield component from the component palette next to the label component.
19. Go to Source tab.
20. Provide the following as element inside the textfield tag

```
<regex>^[\\w\\. -]{1,}\\@([\\da-zA-Z-]{1,}\\.){1,}[\\da-zA-Z-]+</regex>
```

NOTE: This regular expression evaluates email-ids
21. Go to UI tab and select the textfield component.
22. Provide a text of your choice as value for **validation-message** attribute using the properties view.
23. Save the file, reload qafe and invoke the application and test.
 - **NOTE:** In the textfield, providing any arbitrary text other than one with a valid email format will show a dialog with the given validation message.

Following steps will use a less css file to apply styles on components.

24. Create a css folder inside the src\main\webapp directory within the project.
25. Copy and paste qafe.less file into css folder.
26. Provide the following lines of code inside the presentation-tier tag in the qaml source.

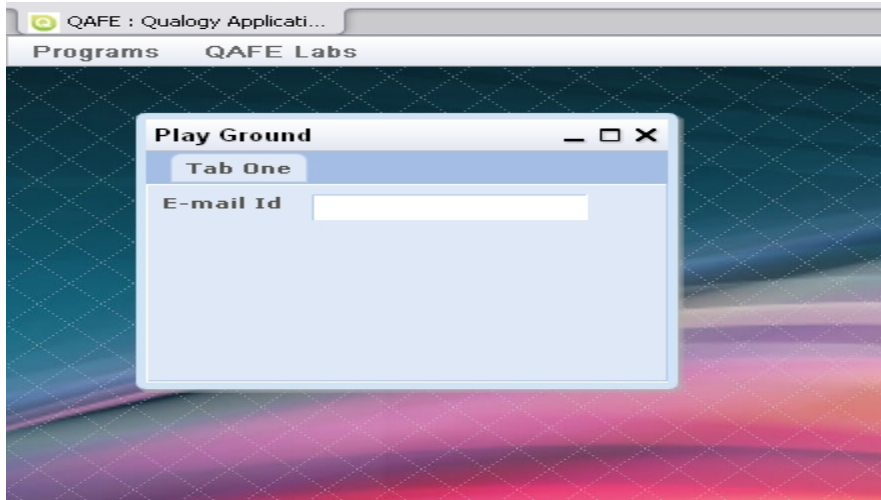
```
<styles>  
  <style location="../../css/qafe.less" window-id="*" />  
</styles>
```

27. Save the file, reload qafe, then refresh the browser and invoke the application to see the style effect.

After exercise 2 the UI tab in eclipse will look as shown below:



Resulting Application:



Exercise 3: Getting familiar with if built-in

This exercise will help to understand the usage of if built-in along with choice component.

The main focus of this exercise is to

- Know how to create and manage an if clause in a QAML application
- Understanding expression evaluation in an if clause
- Using placeholder built-in tags for maintaining variables
- Using store built-in tags to store a value assigned to a variable with global and local scope with respect to an event

Steps:

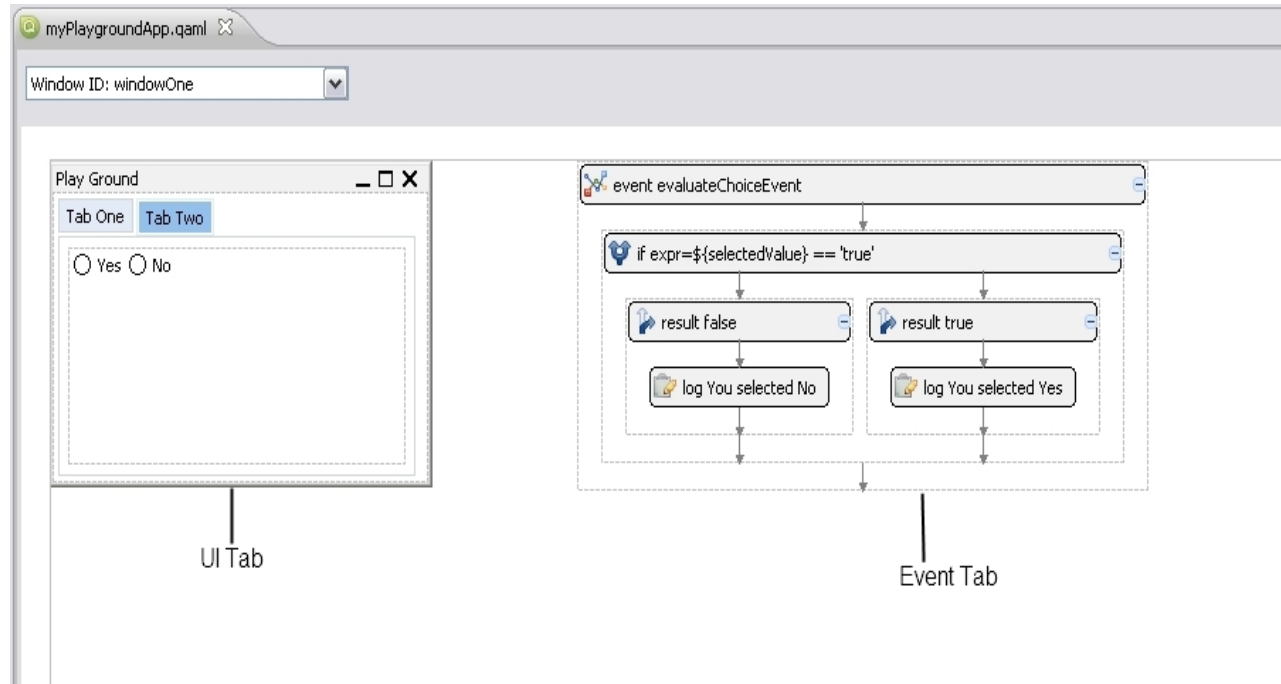
1. Go to the **UI** tab.
2. Drag'n drop a **Tab** component from the component palette into the existing tab panel.
3. Type in *If built-in example* at the prompt which will reflect as value for **displayname**.
4. Drag'n drop a **Choice** component from the component palette into the tab.
 - **NOTE:** The Choice component acts as parent component for Choice-Item(s) in the same way the tab-panel acts as parent container component for tabs..
5. Provide *yesOrNoChoice* as **id** for the **Choice** component.
6. Choose *true* as value for **horizontal** attribute for the **Choice** component.
7. Drag'm drop a **Choice Item** from the component palette and into the choice component.
8. Provide *Yes* as **displayname** and *true* as **value** for the **Choice Item**.
9. Drag'n drop another **Choice Item**, into the choice and provide *No* as **displayname** and *false* as **value** to it.
 - **NOTE:** The value provided for the choice item can be anything that a user wants it to be. This value will be used in expression evaluation part of if built-in.

Following steps creates an onclick event for the choice component and implements logic to evaluate the selected value of the component using QAFE if built-in.

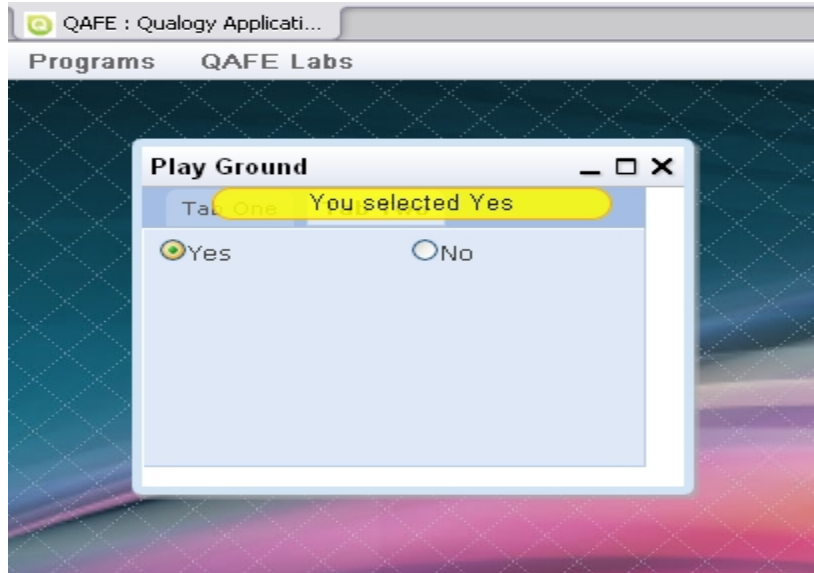
10. Select the **Choice** component from the canvas view.
 - **NOTE:** Make sure that the selection is done on the choice component and not on the choice item.
11. Create an *onclick* event for the **Choice** component using the properties view.

12. Provide *evaluateChoiceEvent* as **id** for the event.
13. Drag'n drop a **If** built-in from the component palette into the created event.
14. Provide *\${selectedValue} == 'true'* for **expression** in the pop-up.
 - **NOTE:** The text that is given as expression will be evaluated using python. So any python script mentioned in the **expr** attribute will be evaluated and there must be a boolean return value for the expression tag to work properly.
15. Enable **result false** check box in the pop-up
16. Click **OK**.
17. Click on the + symbol against the placeholder text in the properties view.
18. Provide *selectedValue* as **name** for the placeholder.
19. Select *component* as **src** for the placeholder.
20. Select *yesOrNoChoice* as **ref** for the placeholder.
21. Click **OK**.
22. Drag'n drop a **Log** built-in from the component palette into the false part of **If** built in.
23. Click in the textfield to provide value for **message** attribute for the Log built-in. A pop-up will appear.
24. Provide *You selected No* as **value**.
25. Click **OK**
26. Drag'n drop a **Log** built-in again into the true part of the If built in.
27. Click in the textfield to provide value for **message** attribute for the Log built-in. A pop-up will appear.
28. Provide *You selected Yes* as **value**.
29. Click **OK**.
30. Save the file, reload qafe and invoke the application again to test the changes made.

After exercise 3 the UI and Event tab in eclipse will look as shown below:



Resulting Application:




Exercise 4: Creating a master-detail application

Following exercise will take you through steps for creating a master-detail application using the drag-'n-drop feature of Qaml builder.

The main focus of this exercise is to

- Understand how to use existing Data Source Explorer feature of eclipse.
- Understand the wizard flow during master-detail application creation.
- Different tiers of Qafe obtained as a result of auto generated qaml code.
- Understand data flow through different tiers.
- Speciality of statements file.
- How to handle exceptions with ORA code.

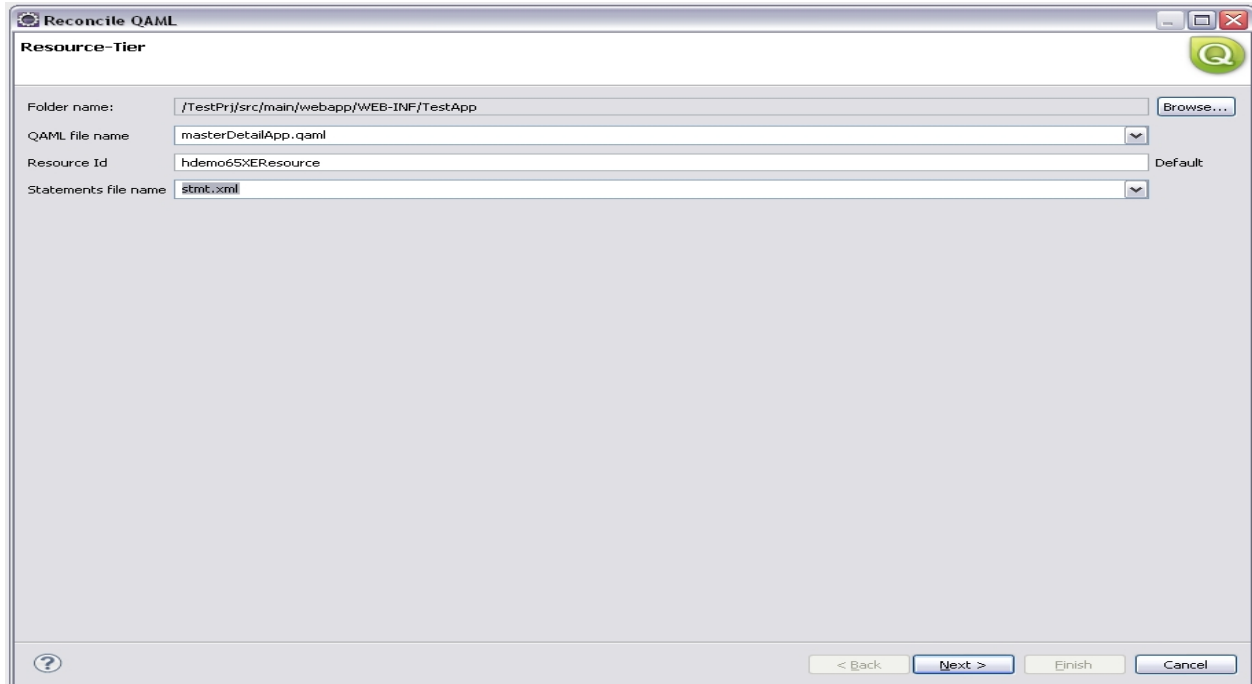
In order to have access to database using eclipse do the following.

1. In **Eclipse > Window > Show View > Data Source Explorer**
 - a. **Note:** If the option is not readily available choose Other > Data Management > Data Source Explorer
2. In Data Source Explorer tab right click on **Database Connections > New**
3. Choose **Oracle** from the list.
4. Optional: Provide a name for the connection of your choice.
5. Click **Next**.
6. Click the symbol ->  next to the **Drivers** drop down to add a driver.
7. Choose **Oracle Thin Driver** for version **11**.
8. Go to **JAR List** tab.
9. Click **Add JAR/Zip...**
10. Browse and select **ojdbc5-11.1.0.7.0.jar** in **qafe-web-gwt-<version>\WEB-INF\lib**
11. Select existing **ojdbc1.4.jar** and click **Remove JAR/Zip**.
12. Click **OK**
13. In the connection profile window provide **SID** of the database that you want to connect.
14. Provide the ip of the machine where the database resides for **Host** field.

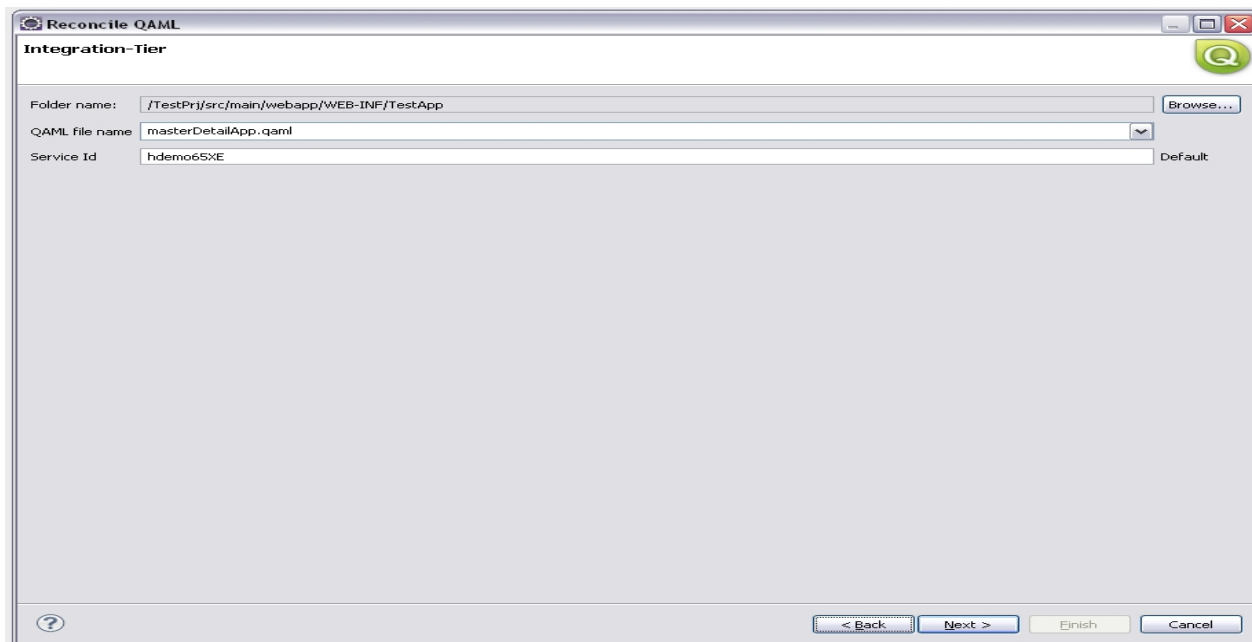
15. Provide **user name** and **password** for accessing the database.
16. Enable **Save Password** check box.
17. Click **Test Connection** to make sure credentials are correct.
18. Click **Next** and then **Finish**

Following steps enables creation of master detail application.

1. In eclipse, go to *SimpleQafeProject* > *src* > *main* > *webapp* > *WEB-INF*.
2. Right click on *SimpleQafeApplication* folder > **New** > **Other** > **QAFE** > **Qaml File**.
3. Click on **Next**.
4. Provide *masterDetailApp* as name for the qaml file.
5. Click on **Finish**. The new qaml file will show up with focus on the **Source** tab.
6. Click on the **UI** tab.
7. Click on the **Window** component in the control bar. A pop-up will appear.
8. Provide *masterDetailWindow* as **id** and *Master Detail Application* as **displayname**.
9. Click on **OK**.
10. Select the new created window by clicking on it.
 - a. **NOTE:** Make sure that the click is performed on the title area of the window. Clicking inside the window will make the rootpanel component selected.
11. Optional: Drag and re-size the window to have a proportionate size with that of the canvas.
12. Use the previously created database connection in Data Source Explorer and open up the nodes till the tables are visible in the QAFE schema.
13. Select Departments and Employees table from QAFE Schema. (You can choose any 2 tables with master detail relationship. Table names specified here can be considered as examples to illustrate the exercise.)
14. Drag'n drop the selected tables into the window created in the masterDetailApplication.qaml.
15. First screen of wizard will come up. Provide a name of your choice for the statements file with .xml as extension.
 - a. **NOTE:** There will be default value for ids which will be used to access the resource and services specified in the integration-tier



16. Click Next. You can opt to change the Service Id



17. Click Next.

At this point the wizard screen will be showing the selected tables in the left side with the first one as highlighted and the columns of that table in the main part.

18. Against **Columns** click on **All** shown as part of title enlisting the columns.

19. Optional: Displaynames of columns can be enabled to be edited by clicking on the same.

20. Click **Apply** button

Reconcile QAML

Service Methods

No tables selected

☐ [form] DEPARTMENTS
☐ [?] EMPLOYEES

Up
Down
Select All
Deselect All
Duplicate

View: form

Number of rows: 1
Number of columns (layout): 2

Primary Key: DEPT_ID

Master Relation: =

Columns:

All	Name	Displayname
<input checked="" type="checkbox"/>	DEPT_ID	DEPT_ID
<input checked="" type="checkbox"/>	DEPT_NAME	DEPT_NAME
<input checked="" type="checkbox"/>	DEPT_LOCATION	DEPT_LOCATION

Up
Down
Apply

< Back Next > Finish Cancel

21. Select Employees table on the left side.
22. Select *List* as **View** from the drop down.
23. In the Master Relation section,
 - a. Select EMP_DEPT for the first drop down.
 - b. Select DEPARTMENTS in the second drop down.
 - c. Select DEPT_ID in the third drop down
24. Click on **All** in the columns list.
25. Optional: Change the display name as per your choice.
26. Optional: Choose the columns that you want to be appearing as part of overflow.
27. Click **Apply**
28. Select both the tables in the left menu.

Reconcile QAML

Service Methods

☒ [form] DEPARTMENTS
☒ [list] DetailEMPLOYEES

Up
 Down
 Select All
 Deselect All
 Duplicate

View: list
 Number of rows: 10
 Number of columns (layout): 2
 Primary Key: EMP_ID

Master Relation: EMP_DEPT = DEPARTMENTS DEPT_ID

All	Name	Displayname	Overflow
<input checked="" type="checkbox"/>	EMP_ID	EMP_ID	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_NAME	EMP_NAME	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_ADDRESS	EMP_ADDRESS	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_EMAIL	EMP_EMAIL	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_PHONE_NUMBER	EMP_PHONE_NUMBER	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_START_DATE	EMP_START_DATE	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	EMP_DEPT	EMP_DEPT	<input checked="" type="checkbox"/>

Up
 Down
 Apply

? < Back Next > Finish Cancel

29. Click Finish.

30. Reload qafe and invoke the master detail application and test.

Following steps focuses on deliberately causing a scenario which causes and exception and then using error-handler built-in of QAFE to manage the exception with ORA code.

31. Go to UI tab.

32. Select the **textfield** component representing EMP_ID column

33. Using the properties view change the **type** attribute to *text*

34. Save and reload the application.

35. In the invoked application in the browser, select an employee from the list and edit the EMP_ID to have a text.

36. Click on Save button. It will show an exception in a dialog box.

37. Note the ORA code shown as part of the exception.

38. Go to the Source tab and add the following code in integration-tier

```
<errors>
    <error exception="ORA-01722" id="invalidNumberInput"/>
</errors>
```

39. Go to the UI tab.

40. Select the Save button component above the employee details part

41. Use the events part in the properties view to find the onclick event on the button.

42. Double click on the event and the Event tab opens up in the canvas view.

43. Drag'n drop an **Error-Handler** built-in from the component palette after the if build-in present in the body of the onclick event.

44. Provide *invalidNumberInputErrorHandler* as **id** for the **Error-Handler**

45. Drag'n drop a **Log** built-in from the component palette into the Error-Handler built-in.

46. Provide a valid text as **value** for **message** for the log built-in.

47. Save the application, reload qafe and test the application with the scenario causing the exception.

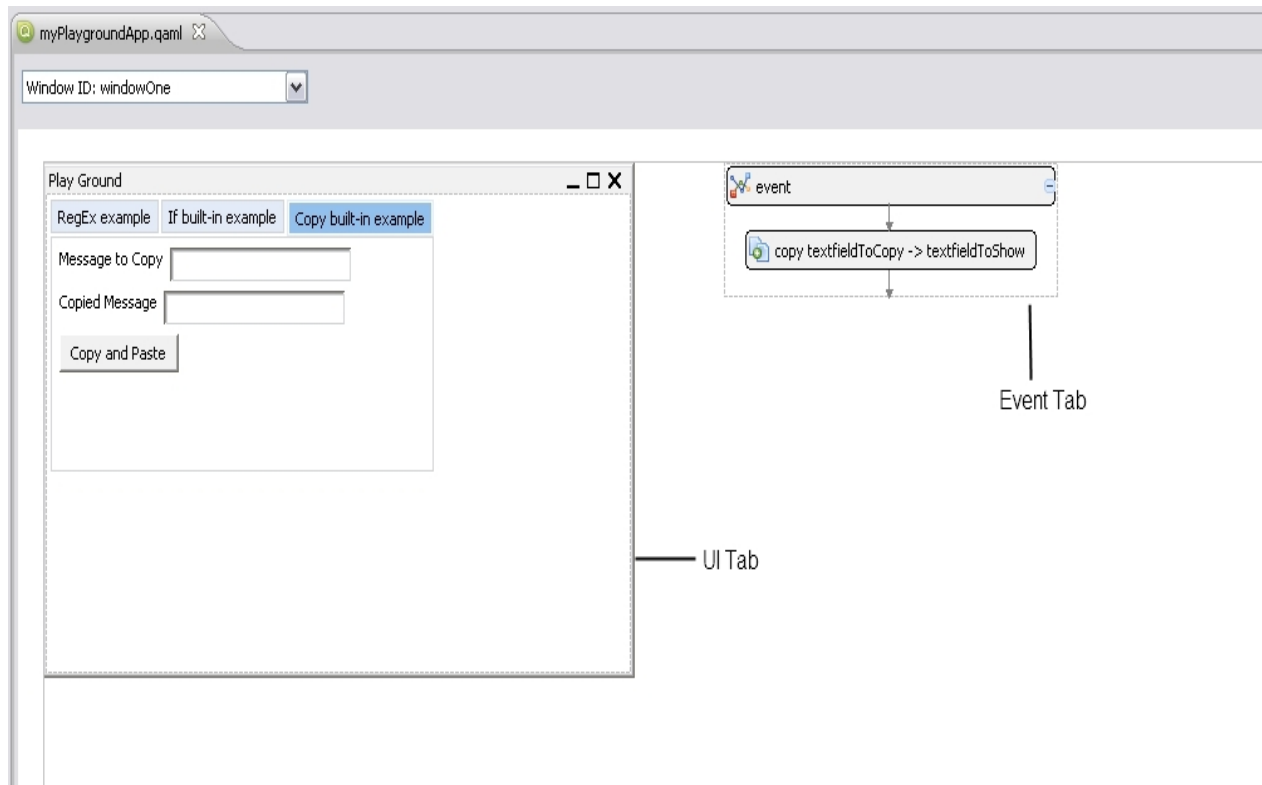
Exercise 5: Copy built-in

Steps:

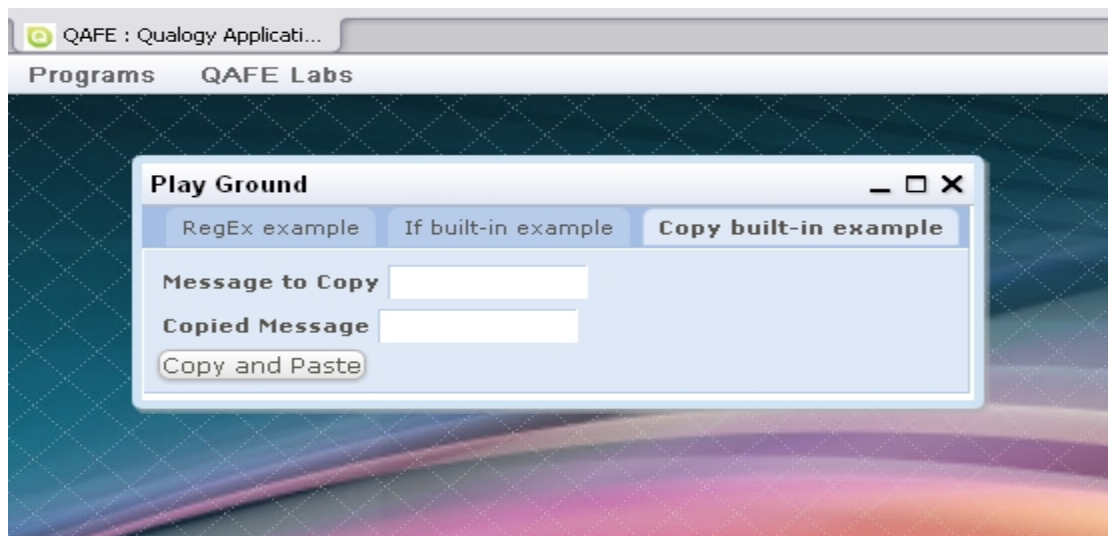
1. Go to the UI tab of myPlaygroundApp.qaml file
2. Add another **Tab** component to the existing tab panel.
3. Type in *Copy built-in example* at the prompt which will reflect as value for **displayname**.
4. Optional: Drag and resize the the window.
5. Drag'n drop a **Textfield** component into the tabpanel.
6. Type *Message to Copy* at the prompt. Hit Enter.
 - o **NOTE:** This will reflect as text for the **displayname** attribute of the text field.
7. Using the properties view provide *textfieldToCopy* as **id** for the text field.
8. Create another text field component after the previously created one.
9. Type in *Copied Message* at the prompt.
10. Provide *textfieldToShow* as **id** for the selected text field component.
11. Set the **editable** attribute of the text field to *false*.
 - o **NOTE:** The difference between the editable and a the disabled attribute for a component is that when the component is disabled, events won't be triggered even if there are listeners implemented for it.Editable attributes determines if input can be received on the component or not
12. Select the tab in the window.
13. Use the Properties view and change the **layout** attribute value to *verticallayout* using the drop down.
14. Create a **Button** component and place it after the text field component.
15. Provide *Copy and Paste* as **displayname** for the button.
16. Provide *copyAndPasteButton* as **id** for the button.
17. The following steps will create an onclick event on the button component. The onclick will use copy built-in to pick up data from one component and to put it into another.
18. Click on **Events** in eclipse Properties view.
19. From the **Listener Type** drop down choose *onclick*.
20. Click **Create Event**. This will take the focus to the Events tab. The new created event will appear in the canvas view.
21. Provide *eventOnCopyAndPasteButton* as **id** for the event.

22. Drag'n drop a **Copy** built in from the built-ins palette into the event.
23. For the **from**, select *textfieldToCopy* from the drop down.
24. For the **to**, select *textfieldToShow* from the drop down.
 - **NOTE:** Notice that the components' id are automatically available in the drop downs
25. Click on **OK**.
26. Save the file, reload the browser and invoke the application and test.

After above exercise the UI and Event tab in eclipse will look as shown below:



Resulting Application:



Exercise 6: Dynamically setting data and visual property of a component

Apart from knowing few more components, the main focus of this exercise is to understand

- How to set data to a component
- How to manage or change visual behavioral property of a component using toggle built-in

Steps:

Following steps focus on how to set data to a component.

1. Go to the **UI** tab.
2. Drag'n drop a **Tab** component from the component palette into the existing tab panel.
3. Type in *Tab Four* at the prompt which will reflect as value for **displayname** attribute of the tab.
4. Drag'n drop a **StackPanel** component into the newly created tab. Re-size if needed
5. Drag'n drop a **Stack** component into the **StackPanel**.
6. Provide *Stack One* as **displayname** at the prompt.
7. Drag'n drop a **Panel** component from the component palette into the **Stack**.
 - **NOTE:** The reason to add a panel inside the stack is to hold more than one component as part of stack panel. If a panel is not added to the stack, then the stack can hold only one component. (When the panel is added, its not necessary to provide displayname for the panel at the prompt. It can be skipped).
8. Optional: Re-size components if needed
9. Drag'n drop a **Slider** component into the created **Panel** and provide *Slider* as **displayname**.
10. Use the properties view and provide *mySlider* as **id**.
11. Provide *100* as value for **max-ticks** attribute.
12. Provide *0* as value for **min-ticks** attribute.
13. Provide *10* as value for **tick-size** attribute.
14. Provide *300* as **width** and *40* as **height** for the slider component.
15. Drag'n drop a **Label** component after the **Slider**.
16. Provide *Current selected slider value:* as **displayname** for the label component.
17. Drag'n drop another **Label** component after the previously created label.
18. Provide *selectedSliderValue* as **id** for the newly created label using the properties view.

The following steps creates an onchange event on the slider component. A set built-in will be used as part of the event to pick up the selected slider component value and show it as a label

19. In the canvas view select the **Slider** component.
20. Click on the **Events** view within the components' properties view.
21. Select *onchange* from the drop down for **Listener Type** and click on **Create Event**.
22. Provide *sliderOnChangeEvent* as **id** for the event created.
23. Drag'n drop a **Set** built-in from the component palette into the newly created event.
24. In the properties view select *selectedSliderValue* from the drop down as **component-id**.
 - **NOTE:** In a **Set** built-in, **component-id** or **name** attribute represents the component on to which a value has to be set. In other words **component-id** or **name** represents the destination component.
25. Select *component* as value for the **src** attribute of the **Set** built-in.
 - **NOTE:** **src** attribute represents the source of data/value that needs to be set to another component. Data to be set can be obtained from another *component*, *pipe* - which is local

storage area of an event, *user* - which is global storage area of events and *message* which takes value for a key from a properties file. eg; A file representing multi lingual data.

26. Select *mySlider* as value for **ref** attribute of the Set.

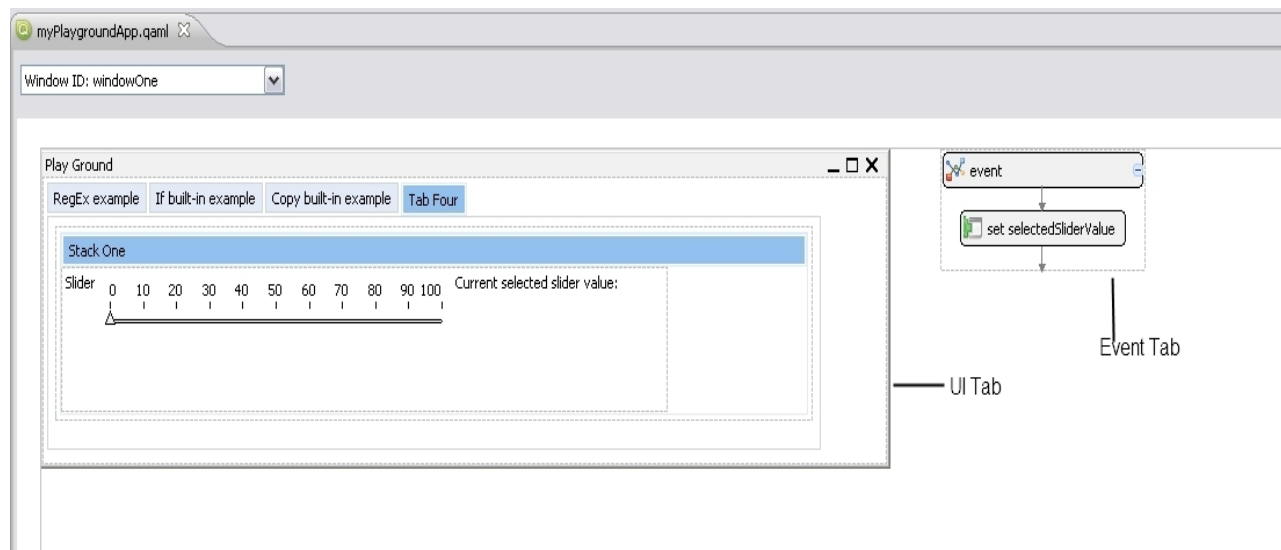
- **NOTE:** ref attribute represents the **id** of the source from where the data is obtained.

27. Save the file, reload the browser and invoke the application again to test the changes made.

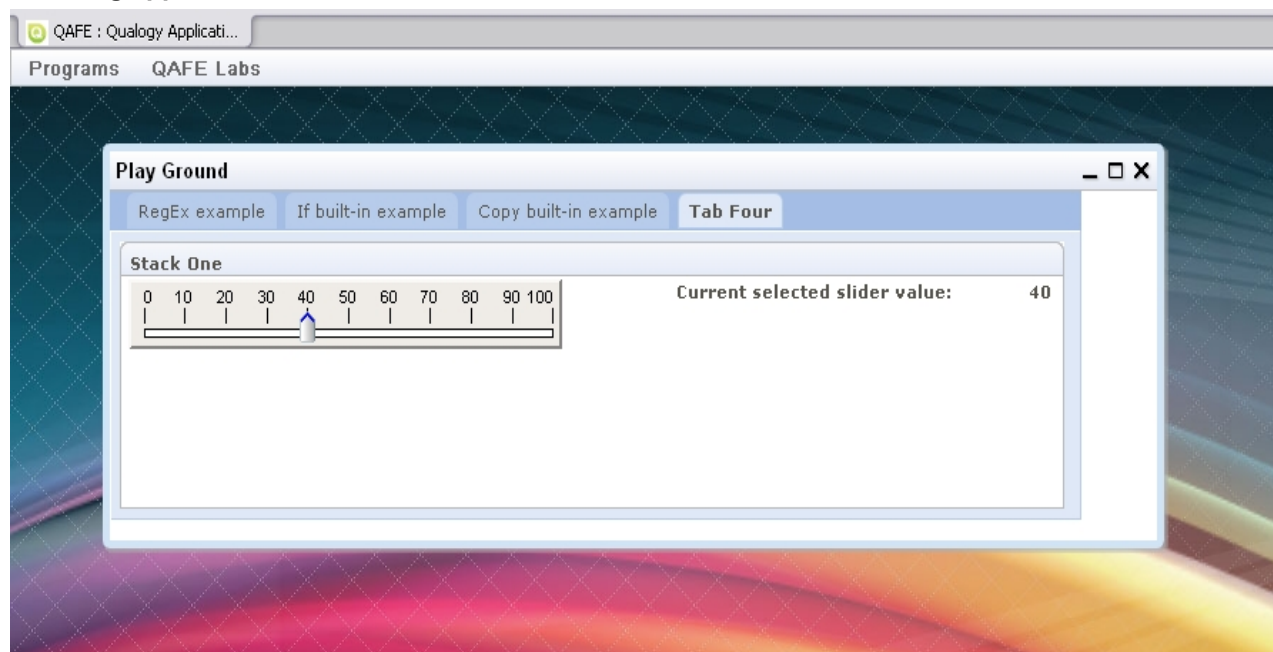
Things to understand from this exercise:

- How to use set built-in to assign data to a component.
- How to create and manage a stack panel.

The result until now of the UI tab and the event tab in eclipse will look as shown below.



Resulting application:



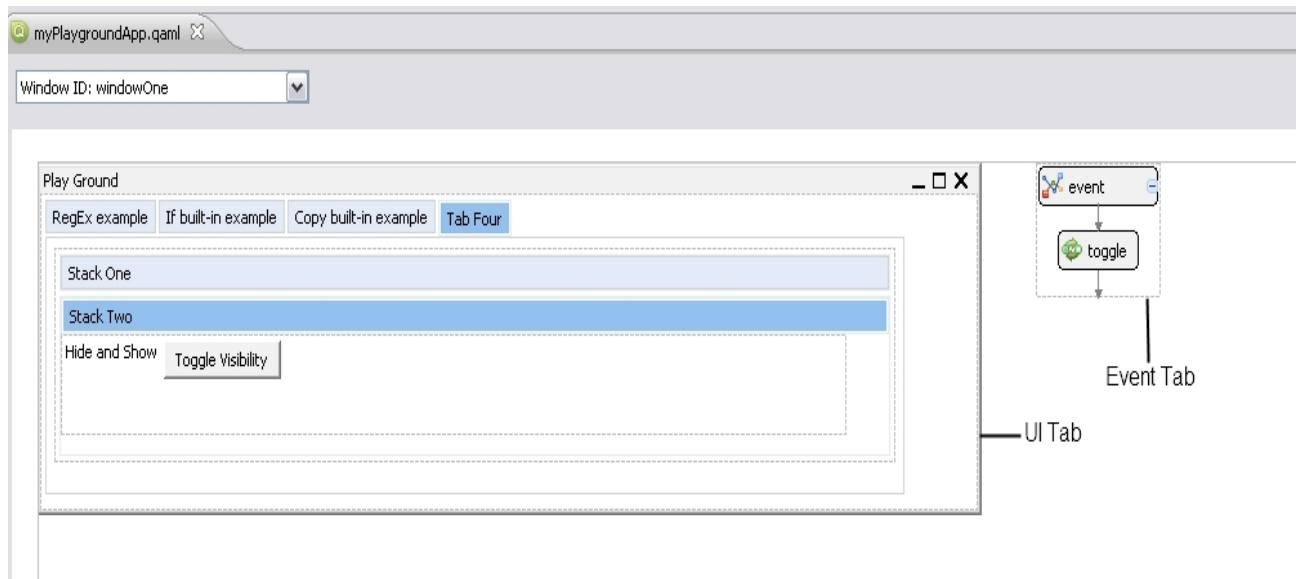
Following steps focus on how manage or change visual behavioral property of a component

1. Go to the **UI** tab.
2. Drag'n drop a **Stack** component after the previously created stack component.
3. Provide *Stack Two* as **displayname**.
4. Drag'n drop a **Panel** component into the stack
5. Drag'n drop a **Label** component into the panel.
6. Provide *Hide and Show* at prompt as **displayname**.
7. Provide *hideAndShowLabel* as **id** for the **Label** component.
8. Drag'n drop a **Button** component after the **Label** in the panel.
9. Provide *Toggle Visibility* as **displayname** at the prompt.
10. Provide *toggleButton* as **id** attribute for the button using the properties view.

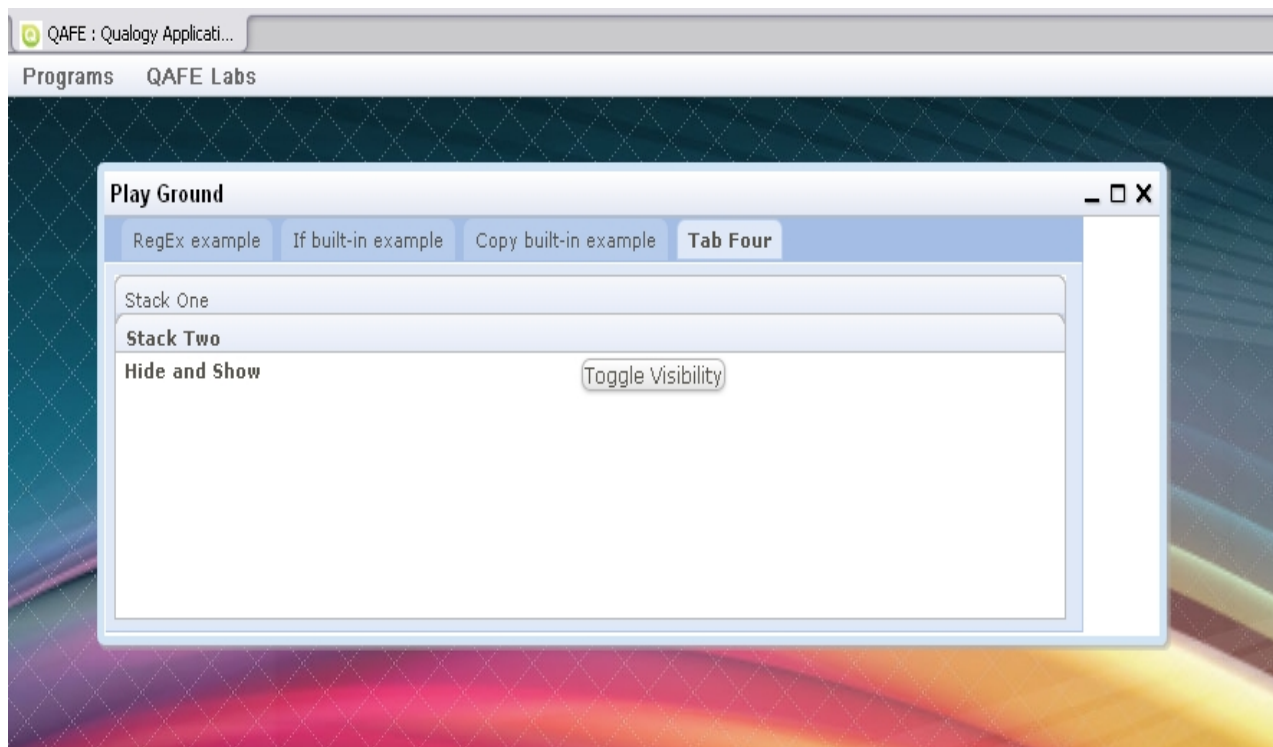
Following steps creates *onclick* events for the button. Toggle built-in will be used to manage the visible attribute of the label

11. Select the button component from the canvas view.
12. Using the properties view create and *onclick* event for the selected button.
13. Provide *toggleVisibilityEvent* as **id** for the event.
14. Drag'n drop a **Toggle** built-in from the component palette into the event
15. Click on the **+** symbol against **component** text to add component(s) for which the toggle built-in should be effective.
 - o **NOTE:** More than one components can be set to have a specific property mentioned in toggle.
16. Choose *ref* as value from the drop down for **component** attribute in the pop-up.
 - o **NOTE:** This represents that we are going to use id of a component to set the property.
17. Choose *hideAndShowLabel* as **value** in the pop-up.
18. Click **OK**.
19. Save the file, reload qafe and invoke the application again to test the changes made.

The result of the UI tab and the event tab in eclipse will look as shown below.



Resulting application:



Exercise 7: Dynamically changing style of a component

Focus of this exercise is to understand how to use change-style built-in in Qafe to manipulate appearance of a component dynamically

Steps:

Following steps focus on how to set data to a component.

1. Go to the **UI** tab.
2. Drag'n drop a **Tab** component from the component palette into the existing tab panel.
3. Type in *Tab Five* at the prompt which will reflect as value for **displayname** attribute of the tab.
4. Drag'n drop a **Label** component into the newly created tab.
5. Provide a text for *displayname* at the prompt.
6. Provide an *id* for the label component using the properties view.
7. Using the Events option in the Properties view create a *onmouse-enter* event on the label.
8. In the **Event** tab, Select a **Change Style** built-in and place it in the event.
9. In the properties view for the **Change Style** built-in, Click on the add symbol against **style-action**.
10. Provide **key** as *color* and **value** as *red* in the pop-up and click Ok..
11. Click the add symbol against **component**.
12. Select *ref* from the dropdown for **component**
13. Select the id of the label component provided earlier as **value** in the pop-up and click Ok.
14. Save the file and go to **UI** tab.
15. Select the label from the canvas view and create an *onmouse-exit* event using the Events option in Properties view.
16. Create a change-style built-n as done in the previous step and provide *blue* as **value** for the **key** *color*.
17. Save, reload and test the application.

Java class

Find you entries in SQL Statement